U.S. PATENT APPLICATION FOR A SYSTEM AND METHOD FOR GENERATING 128-BIT CYCLIC REDUNDANCY CHECK VALUES WITH 32-BIT GRANULARITY

ASSIGNEE: iReady Corporation

Attorney Docket No. SPL3004 – P2

Stafford Partners, LLC
40 Lake Bellevue, Suite 100
Bellevue, WA 98005

SYSTEM AND METHOD FOR GENERATING 128-BIT CYCLIC REDUNDANCY CHECK VALUES WITH 32-BIT GRANULARITY

FIELD OF THE INVENTION

This invention relates to the field of error detection in data transmission and more particularly to a System and Method for Generating Cyclic Redundancy Check (CRC) values in a system having a data bus adapted simultaneously handling a plurality of blocks in parallel.

BACKGROUND OF THE INVENTION

As the internet and computer networking continue to evolve, data transmission speeds are increasing as well as the volume of data transmitted. The increase in data traffic is occurring in Local Area Networks (LANs) based on Ethernet and other transport mechanisms such as Wide Area Networks (WANs), and Storage Area Networks (SANs) which could use Ethernet or any of a number of data transport mechanisms. Similarly, the amount of data moving through Internet Protocol (IP) based networks such as the internet continues to grow substantially.

Accordingly, users face a growing need for new ways to store and maintain their data. Today's technology offers three basic storage options: Direct Attached Storage (DAS), Network Attached Storage (NAS) and Storage Area Networks (SAN).

In its most basic form, Direct Attached Storage consists of a disk drive directly attached to a personal computer or server. One of the most common methods of transferring data between a hard drive and its associated personal computer or server is the Small Computer Systems Interface (SCSI). Other methods, such as SATA and IDE are well known.

The SCSI protocol uses commands to transfer blocks of data, which are low level, granular units used by storage devices, as opposed to LANs, which typically use file based methods for transferring data. The overall operation and an architectural

description of the SCSI protocol is available from the American National Standards Institute (ANSI), the specific specification having the designation ANSI/INCITS 366-2003, titled *Information Technology – SCSI Architecture Model – 2 (SAM-2)*, herein incorporated reference, and herein referred to as the SCSI Specification.

As internet traffic and storage needs have grown, there is a growing convergence between storage devices, protocols, and IP based transport mechanisms. For example, current SCSI storage devices are designed to work over a parallel cable having a maximum cable length of 12 meters, While IP based transport mechanisms have no data transmission distance limitation.

At the present time, the storage industry and the various industry entities responsible for developing and maintaining the various Internet Protocols are working together to develop standards to enable SCSI based data transfers over the internet. Specifically, the IP Storage (IPS) Working Group of the Internet Engineering Task Force (IEF) is in the process of finalizing a specification for encapsulating SCSI commands in the known TCP/IP protocol. The Internet SCSI (iSCSI) protocol for block storage is predicated on standard Ethernet transports. The iSCSI protocol defines the rules and processes to transmit and receive block storage data over TCP/IP networks, both of which employ error detection techniques to ensure data integrity during transmission.

iSCSI replaces the parallel SCSI direct cabling scheme with a network fabric. iSCSI is transport independent and will support any media that supports TCP/IP. Servers and storage devices that support iSCSI connect directly to an existing IP switch and router infrastructure. iSCSI enables SCSI-3 commands to be encapsulated in TCP packets and delivered reliability over IP networks. The iSCSI specification is complete and undergoing final ratification within the IETF. The current iSCSI specification is available from the IETF under the designation *draft-ietf-ips-iscsi-20.txt*, dated January 19, 2003, and herein referred to as the iSCSI Specification. iSCSI network interfaces under development will be capable of transferring data over the internet in speeds approaching 20 Gbits/sec. The iSCSI protocol is just one example of a network storage protocol, which may employ the System and Method of the present invention, although those skilled in the art will appreciate that the System and Method of the present

invention is useful in any type of data transfer protocol where CRC checksums are useful or required.

Cyclic Redundancy Check (CRC) techniques are used to verify the validity of data contained in data blocks or data segments transferred between devices, such as a storage device and a computer or server system, as described above. In other applications, CRC techniques may be used in data communication systems to verify the validity of data blocks transmitted between devices that are geographically distributed, such as those which employ the Internet as a transmission media, or in systems where data is transmitted over conventional telephone lines using modems.

A variety of CRC techniques are well known and have become widely deployed because of their highly reliable error detection capabilities, speed and relative ease of implementation. For example, a description of CRC applications may be found in the text: Press, W. H.; Flannery, B. P.; Teukolsky, S. A.; and Vetterling, W. T. "Cyclic Redundancy and Other Checksums." Ch. 20.3 in *Numerical Recipies in Fortran: the Art of Scientific Computing, 2nd ed.*, Cambridge, England: Cambridge University Press, pp. 888-895, 1992. A further example of a 32-bit CRC application is described in the industry standard publication designated: IEEE 802.3, 2000 Edition, CSMA/CD (ISO/IEC 8802-3:2000(E), which describes the requirements for what is commonly referred to as "Fast Ethernet".

CRC values can be appended to a variety of data types ranging from blocks of data, having a predefined number of bits, to predefined groups of blocks, known as segments. CRC values are determined according to a predetermined algorithm which is known in both an initiator system and a target system. The actual CRC value is typically a function of the specific algorithm used, the actual data communicated, as well as the CRC value used in a prior communication session.

CRC algorithms can vary in complexity from those based on simple binary arithmetic to more complex algorithms using polynomials and some form of binary division or multiplication. The overall reliability of any given CRC algorithm is a function of the complexity of the algorithm used as well as the number of bits used in calculating the CRC value. For example, these algorithms may not detect an error for an N-bit checksum, because there is a possibility that $1/2^N$ of random blocks will have the

same checksum for inequivalent data blocks. However, as the value of N increases, the probability that two inequivalent blocks will have the same CRC value decreases. If N is sufficiently large, the probability of a detecting an error drops to a statistically insignificant value. Further, algorithms based on polynomial division or multiplication are less susceptible to error than those based on simple addition or subtraction.

In general, CRC values are generated using binary or Modulo-2 arithmetic. Therefore, the multiplication process used to generate a CRC value in a complex polynomial is merely a sequence of logical ANDs and XORs which can be readily implemented with well known hardware or software techniques. To detect errors in any given data communication, a CRC value is calculated prior to data transmission and appended to a data packet or segment. When the data packet or segment is received, the CRC value is recalculated, and compared to the CRC value appended to the data packet or segment. If the two CRC values match, no errors occurred during data transmission. If the two CRC values do not match, an error has occurred, and re-transmission is required.

SUMMARY OF THE INVENTION

A System and Method for generating Cyclic Redundancy Check (CRC) values in a system for adapted simultaneously handling a plurality of blocks in parallel is described. Included is a memory or other storage device for storing data blocks, wherein the memory or other storage device is adapted to output a plurality of data blocks simultaneously. A data bus, coupled to the memory, provides a data path wide enough to accommodate the parallel data blocks and is further coupled to a plurality of CRC cores, wherein each CRC core calculates a CRC value for every combination of valid data blocks on the data bus. A multiplexer coupled to the CRC cores selects the output of one of the CRC cores based on the number of valid data blocks on the data bus. The selected CRC value is then appended to a data segment for transmission to another device.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a diagram showing the protocol stack in a typical Data Network System.

Figure 2 is a diagram of a data structure of typical TCP/IP Data Communication Packet, which includes an appended CRC value.

Figure 3 is a system diagram a Data Transmission System which is adapted for use in accordance with the principles of the Present Invention.

Figure 4 is a schematic diagram of the CRC Module of Figure 3.

DETAILED DESCRIPTION OF THE INVENTION

The System and Method for Marker insertion of the present invention is useful in data transmission systems such as those based on the TCP/IP protocol. Figure 1 shows a Data Network 100, which may employ standard networking protocols such as TCP/IP as well as a storage protocol such as SCSI. The Data Network 100 comprises and Initiator System 102 and a Target System 104. The Initiator System 102 includes a Physical Data Link 106, which provides a physical connection to the Internet 108 via any type of physical connection, such as an Ethernet connection common in most Local Area Networks. The Physical Data Link 106 is coupled to a Network Stack 110, which exchanges data with the Physical Data Link 106 in accordance with a Network Communication Protocol such as TCP/IP. The Network Stack 110 is further coupled to a Storage Protocol Services Processor 112 that exchanges data with Network Stack 110. The Storage Protocol Processor 112 processes requests from a Storage Application 114 and encapsulates or decodes packets as requested by Storage Application 114 in accordance with a predetermined data storage protocol such as SCSI.

The Target System 104 includes a set of components that complement those of the Initiator System 102. Specifically, the Target System 104 comprises a Physical Data Link 116, a Network Stack 118, a Storage Protocol Services Processor 120 and a Storage Device Server 122, wherein each of the respective devices in Data Network 100 at each layer are in logical communication with each other. For example, each of the respective Network Stacks 110, 118 are in cooperative communication through the Physical Data Links 106, 116 to establish and maintain network connections, via a Network Communication Protocol such as TCP/IP over the Internet 108, by addressing the appropriate target and destination IP addresses, and opening ports and sockets during an active connection. Similarly, the respective Storage Protocol Services Processors 112, 120 are in logical communication with each other in establishing connections, negotiating parameters and exchanging Data Communication Packets such as those specified in the

iSCSI specification. Finally, the Storage Application 114 is in logical communication with the target Storage Device Server 122 in the exchange of data blocks, such as those defined in the SCSI specification.

In operation, the respective Initiator and Target systems 102, 104 operate as typical host and storage devices that are logically coupled with a network connection and through the various service and transport layers below. Thus, any distance limitations imposed by the physical characteristics of the directly connected storage interfaces are eliminated. Further, in many network configurations, Personal Computers, Servers and various Network Attached storage devices will include complementary Target and Initiator Systems. However, the present invention is particularly useful in the context of one device initiating a data communication session with another.

Figure 2 shows an exemplary Data Communication Packet 200 for transmission via TCP/IP according to a network storage protocol such as the one described in the iSCSI Specification. As shown, the Data Communication Packet 200 includes an IP Header 202 and a TCP header 206 which are defined in accordance with the industry standard TCP/IP protocol. IP and TCP headers are used in establishing connections and include parameters such as a source address, destination address, and port identification. The TCP/IP protocol also provides for the insertion of an IP checksum 204 between the IP Header 202 and TCP Header 206 that may be used for error detection while establishing a connection. Following the TCP Header 204 are a Storage Protocol Header 208 and Storage Device Commands 212. Data segment 214 comprises a group of data blocks requested from a Storage Application 114, which are read from Storage Device Server 122. A CRC value 216 is appended to the end of Data Communication Packet 200 for error detection. The Storage Protocol Header 208 may include a number of parameters such as frame length, frame identifier, communication session options or other desired information. The storage device commands may include standard commands such as those used in directly attached SCSI systems.

As will be described in greater detail below, the CRC value 216 is calculated based on a predetermined CRC algorithm as well as the contents of Data Segment 214. Since the Network Storage Protocol Header 208, Storage Device Commands 212, and Data Segment 214 are exchanged between Initiator and Target Systems as data blocks

within a TCP/IP connection, the physical transport layer becomes somewhat irrelevant. The Network Storage Protocol and Storage Device information appear as nothing more than a string of binary values sent over a physical layer. As such, the entire internet infrastructure is available as a physical transport mechanism for data block transfers.

Referring now to Figure 3, a system diagram of Data Transmission System 300 is shown. Those skilled in the art will appreciate that the Data Transmission System 300 may be implemented in any of a number of ways including implementation entirely in software or hardware, or any combination thereof. As data transmission rates continue to increase, it is becoming increasingly difficult for typical Central Processing Units found in Personal Computers and Servers to manage data traffic without having a performance impact on total system performance. Thus, it is becoming increasingly common for data transmission systems such as those based in the iSCSI Specification to be implemented in devices known as Transmission Offload Engines.

An overview of various Transmission Offload schemes is available from the Storage Networking Industry Association (SNIA). For example, a Whitepaper published by the SNIA IP Storage Forum and entitled *iSCSI Building Blocks for IP Storage Networking* discusses various iSCSI implementations and Transmission Offload Engines. The Data Transmission System 300 is suitable for use as an implementation of Target System 104.

In the Data Transmission System 300, a Data Storage module 302 is used to store data in a host system such as a Personal Computer, Server or Network Storage Device and may include one or several hard disk drives or any type of random access memory. The Data Storage Module 302 is coupled to Data Controller 304 with Memory Control Bus 306. Data Storage Module 302 is further coupled to CRC Module 308 through Data Bus 310 and Control Bus 312. Control Bus 312 is used to synchronize transfers of data blocks between the Data Storage Module 302 and CRC Module 308. Control Block 314 is cooperatively coupled to CRC Module 308 with Control Bus 316. Control Block 314 is further coupled to Data Controller 304 with Control Bus 318. The specific operation of the various control busses 306, 312, 316 and 318, Data Controller 304, Control Block 314 and CRC Module 308 is discussed in further detail below.

The output of CRC Module 308 is coupled to the Network Stack 320 with Data Bus 322 for generating a CRC value and appending it to the Data Communication Packet 200 described in conjunction with Figure 2. Once the Data Communication Packet 200 has been aggregated in Network Stack 320, it is then sent to the Physical Data Link 116 via Data Bus 324.

Referring now to Figure 4, the CRC Module 308 is shown in detail. In the preferred embodiment of the present invention, Data Bus 310 is 128 bits wide, and can accommodate four 32-bit data blocks simultaneously. In other words, Data Bus 310 is capable of accommodating data paths of 32-bits, 64-bits, 96-bits or 128 bits, as required during a transfer of data from Data Storage Module 302, as represented by Data Paths 403, 405, 407 and 409, respectively. Typical block based data storage systems, such as the one described in the SCSI specification use a 32-bit wide data path, and data transfers occur on a block-by-block basis. The present invention achieves a substantial improvement in performance over prior implementations by providing for transfers of a plurality of data blocks in parallel. While the preferred embodiment of the present invention is described in the context of a 128-bit wide data bus, those skilled in the art will appreciate that the Data Bus 310 could easily be adapted to accommodate much wider or narrower data paths, as required by a given implementation.

Data Bus 310 is coupled to a plurality of registers 402 – 408, each of which is adapted to temporarily store valid data blocks present on Data Bus 310, in general, and on more specifically on Data Paths 403, 405, 407, and 409, respectively. In the embodiment shown, Data bus 310 is implemented with a 128-bit data bus with Dath Path 409 handles data lines designated <127:0>. Register 402 is physically coupled to Data Path 403 and handles data lines <31:0>; Register 404 is physically coupled to Data Path 405 and handles data lines <63:0>; Register 406 is physically coupled to Data Path 407 and handles data lines <95:0>; and Register 408 is physically coupled to Data Path 409 and handles data lines <127:0>. Therefore, a register is associated with each of a plurality of Data Paths for temporarily storing each possibility of valid 32-bit data blocks on Data Bus 310.

In the course of any given data transmission, the final block of the data transmission may occur on the boundary of any of the above possibilities. For example,

as data blocks are read from Data Storage Module 302, the end of the data transmission may be a single 32-bit block, or may be as many as four 32-bit blocks. Each of the respective Registers 402-408 is coupled to a corresponding CRC core specifically adapted for each possibility of valid data blocks on Data Bus 310. As shown in Figure 4, Register 402 is coupled to 32-bit CRC core 410; Register 404 is coupled to 64-bit CRC core 412; Register 406 is coupled to 96-bit CRC core 414; and Register 408 is coupled to 128-bit CRC core 416. Therefore, a CRC core is provided for every instance or granularity of valid data blocks on Data Bus 310.

CRC module 308 operates in conjunction with memory control signals *dword_valid* and *data_valid* which are generated by Data Controller 304, and control signal *reset_CRC* and *CRC_seed* which are generated within Control Block 314. During a transfer from Data Storage module 302, data blocks are read into the respective registers 402- 408. Control signal *dword_valid* is coupled to 4-1 Multiplexer 418, and is a 2-bit value that indicates which of Registers 402-404 contain valid data blocks. For each data transfer cycle, CRC cores 410-416 calculate CRC candidates for all four possibilities that valid data may be present in Registers 402-408. The correct candidate is selected in Multiplexer 418 based on the value of *dword_valid*, which is calculated with a decrementing counter that is initialized prior to data transfer from Data Storage Module 302, and is decremented based on the number of valid data blocks on Data Bus 310. In any given data transfer, it is assumed that all 128-bits are valid until the last set of data blocks are transferred. The control *CRC_seed* is coupled to 2-1 Multiplexer 420 and is used to promote a CRC seed at the end of each data transmission. When the end of a data transmission is reached, *CRC_Seed* is asserted at Multiplexer 420 so that the current CRC value is fed back to the respective CRC cores for generating a new CRC value for the next data transmission, upon assertion of the control signal *reset_CRC*.

Once a CRC value has been determined, it is latched into Register 422 upon assertion of the control signal *data_valid*, and is output to Data Bus 322 for appending to the a data packet in Network Stack 320.

While the various embodiments described above have been described with reference to Data transmission Systems, it should be understood that they have been presented by way of example only, and not limitation. Thus, the breadth and scope of a

preferred embodiment should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with following claims and their equivalents.